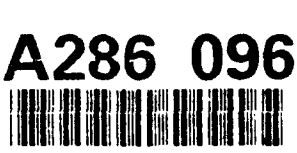
AD-A286 096



(1)

ARMY RESEARCH LABORATORY

Network Distributed Global Memory for Transparent Message Passing on Distributed Networks

Jerry A. Clarke

ARL-CR-173 October 1994

prepared by

Computer Sciences Corporation
3160 Fairview Park Dr.
Falls Church, VA 22042

DTIC
ELECTE
NOV 1 0 1994
D

under contract

DAAL03-89-7C-0088

94-34622

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

94 11 7 088

NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1994	Progress, 1 November 1993–7 March 1994

4. TITLE AND SUBTITLE

Network Distributed Global Memory for Transparent Message Passing on Distributed Networks

5. FUNDING NUMBERS

C: AHPCRC
C: DAAL03-89-7C-0088

6. AUTHOR(S)

Jerry A. Clarke

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computer Sciences Corporation
3160 Fairview Park Dr.
Falls Church, VA 22042

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
ATTN: AMSRL-OP-AP-L
Aberdeen Proving Ground, MD 21005-5066

10. SPONSORING MONITORING AGENCY REPORT NUMBER

ARL-CR-173

11. SUPPLEMENTARY NOTES

Contracting Officer's Representative for this report is Mr. Harold J. Breaux, U.S. Army Research Laboratory, ATTN: AMSRL-CI-A, Aberdeen Proving Ground, MD 21005-5067.

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Distributing compute-intensive applications across multiple platforms can significantly reduce the total execution time of the application. Programming such a system typically involves sending explicit messages between cooperating processes. Keeping track of these messages, however, can be a difficult, error-prone task.

Shared memory models are typically easier to program. Network Distributed Global Memory (NDGM) allows the programmer to view the physically distributed environment as a single, contiguous address space. Instead of sending an explicit message to another node, processes write and read from a global block of memory.

14. SUBJECT TERMS

NDGM: Network Distributed Global Memory, Distributed Computing Computer programs, software (computers), networks

15. NUMBER OF PAGES

17

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and/or Special
A-1

INTENTIONALLY LEFT BLANK.

LIST OF FIGURES

LIST OF TABLES

INTENTIONALLY LEFT BLANK.

1. INTRODUCTION

Network Distributed Global Memory (*NDGM*) is implemented on top of a message passing layer called Message Relay System (*MRS*). *MRS* presents a common interface for sending and receiving messages over several different interfaces: Transport Control Protocol/Internet Protocol (TCP/IP), Shared Memory Arena, Fifos, and "stdio" file pointers. *NDGM* sends messages via *MRS* to copy sections of local memory to and from the global address space.

In addition to data transfer, *NDGM* provides mechanisms for program coordination. These include:

Semaphores: Allow one node exclusive access to some service
Memory Locks: Allow one node exclusive access to an area of memory
Barriers: Allow multiple nodes to block until all have reached a certain point

NDGM is implemented using a client-server model.

MRS allows processes on heterogeneous machines to communicate via message passing. Each *node* in *MRS* can choose from several different physical transport mechanisms for its data. In Figure 1, nodes on the same machine communicate via shared memory, while communication across machines is accomplished via TCP/IP.

In addition to direct message passing, *MRS* allows nodes to send their messages indirectly. For example, node 1 can communicate with node 3 by *relaying* its message through nodes 2 and 4. Also a message may be broadcast to all known nodes. For example, node 1 could send the same message to nodes 2, 3, and 4 by sending one broadcast message to node 2.

The maximum size of the message is determined when the node is created and is application defined. It is also possible to directly access a node's data space in order to minimize buffering.

As shown in Figure 2, an application might define some custom structure to pass as messages that contain an opcode and some arguments. By assigning a structure pointer to the data area of the node, the

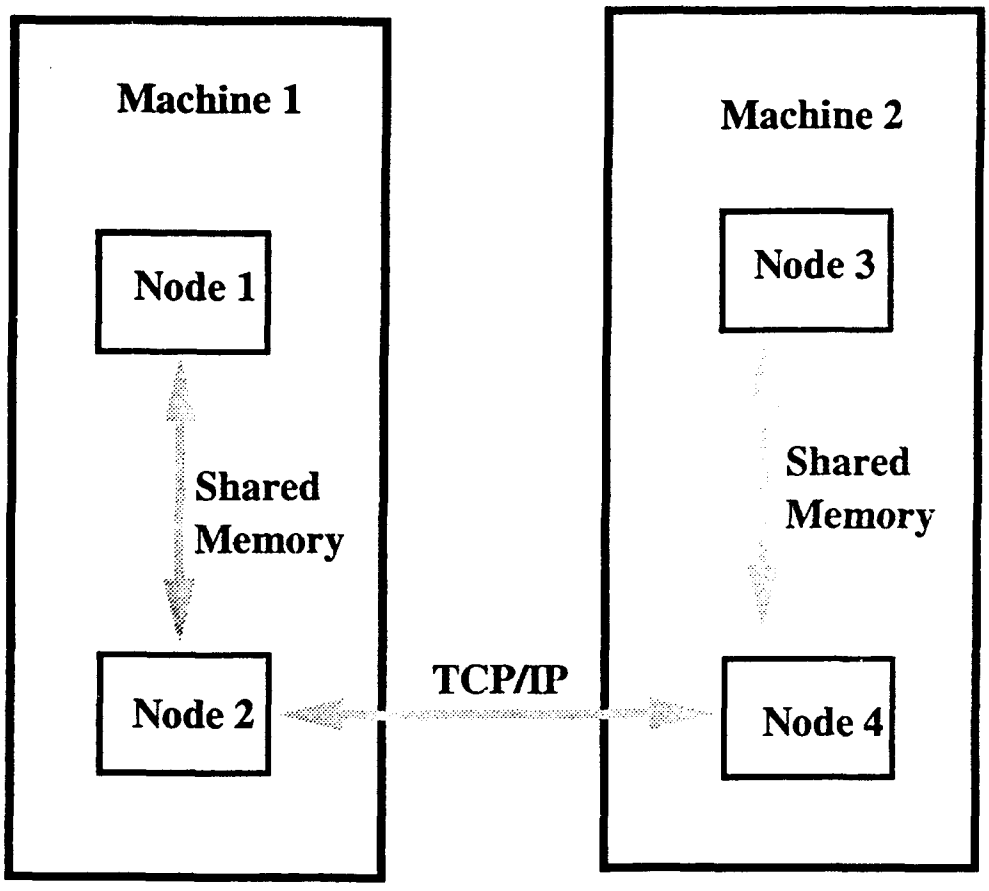


Figure 1. Message relay system basics.

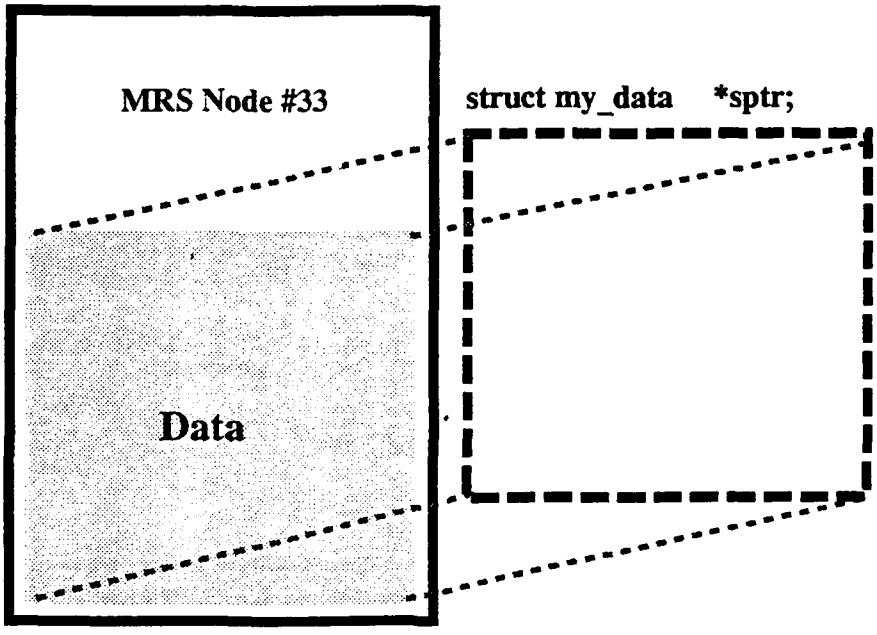


Figure 2. Message relay system node data area.

message data can be accessed as any arbitrary structure. In addition, the data does not have to be copied into the *MRS* message buffer since it is already in the proper position. Such a codelette might look like this:

```
typedef struct {
                int             opcode;
                int             array_size;
                float           data[1];

                } MY_STRUCT;

MY_STRUCT   *sptr;

MRS_NODE                    *node;

/* Open the node */
node = mrs_open(.....);

/* Assign sptr to the node's data */
sptr = MRS_NODE_DATA(node);

/* Fill it up */
for(i=0 ; i < 1000 ; i++){
                sptr->data[i] = 5.0 * i;
                }

mrs_send(node, sizeof(MY_STRUCT) + (1000 * sizeof(float)), (char *)sptr);
```

An *MRS* node, shown in Figure 3, provides a convenient abstraction that allows message passing to be accomplished across several low-level mechanisms. Each node contains a unique ID, the hostname on which it was created, and the type of processor of that machine. The node contains function pointers to routines that provide the actual data transport. Currently, transport is provided for: TCP/IP, a shared memory queue mechanism, a generic memory buffer, and stdio file pointers such as Fifos.

The *MRS* node also points to a message area. This area can be provided by the user, or the *MRS* library will allocate one via *calloc()*. When a message is received, it contains the owner (originator) of the message, the last node to handle the message, and an intended delivery path. If the receiving node can connect to the destination node, the intended list is ignored and the message is delivered directly.

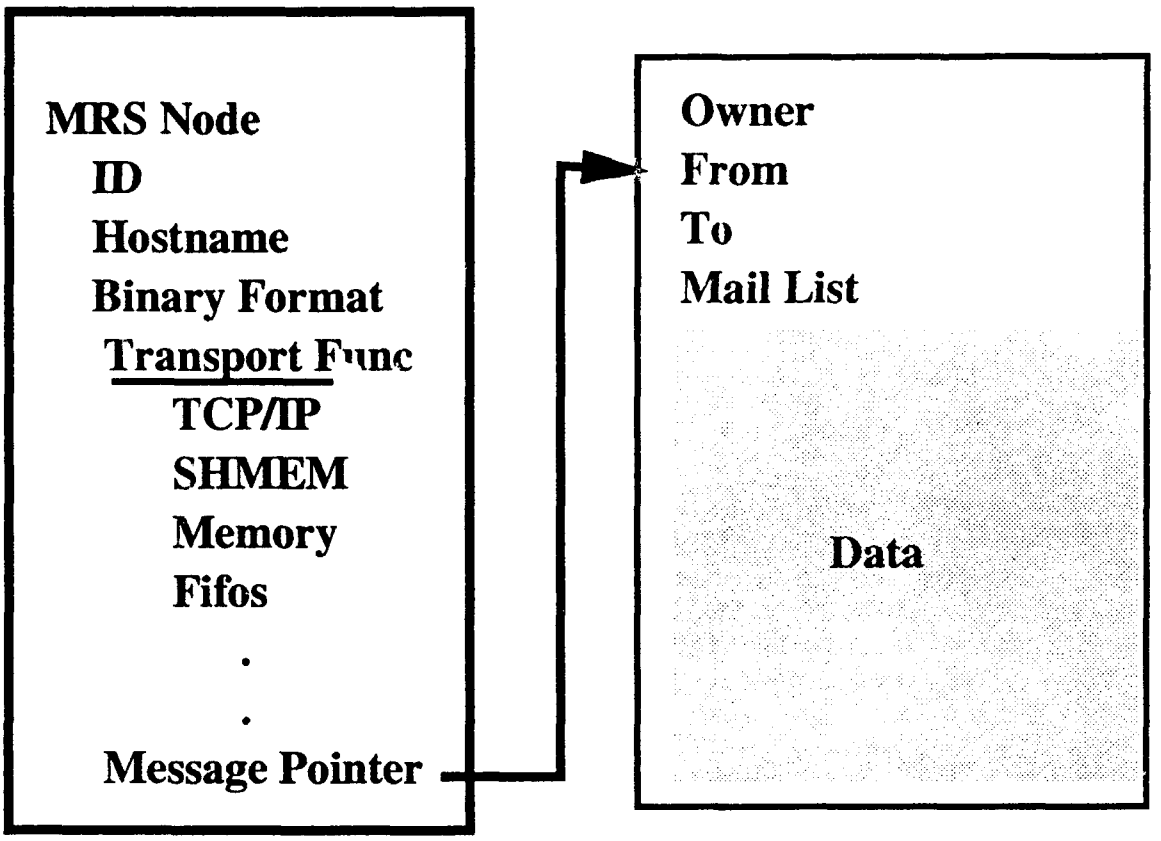


Figure 3. Message relay system node information

There are also "broadcast" messages. These are messages that, once received, are broadcast to all known nodes.

Since the node's data space can be assigned by the application, many nodes can share the same memory area if the application knows that only one node will access the area at a time.

In Figure 4, for example, if an application needed to maintain 70 TCP/IP connections, each capable of a one megabyte message, the application could assign the same data buffer to all of the nodes. As long as data is copied out of the data buffer if needed for later use, it is not necessary to maintain a separate buffer for each node.

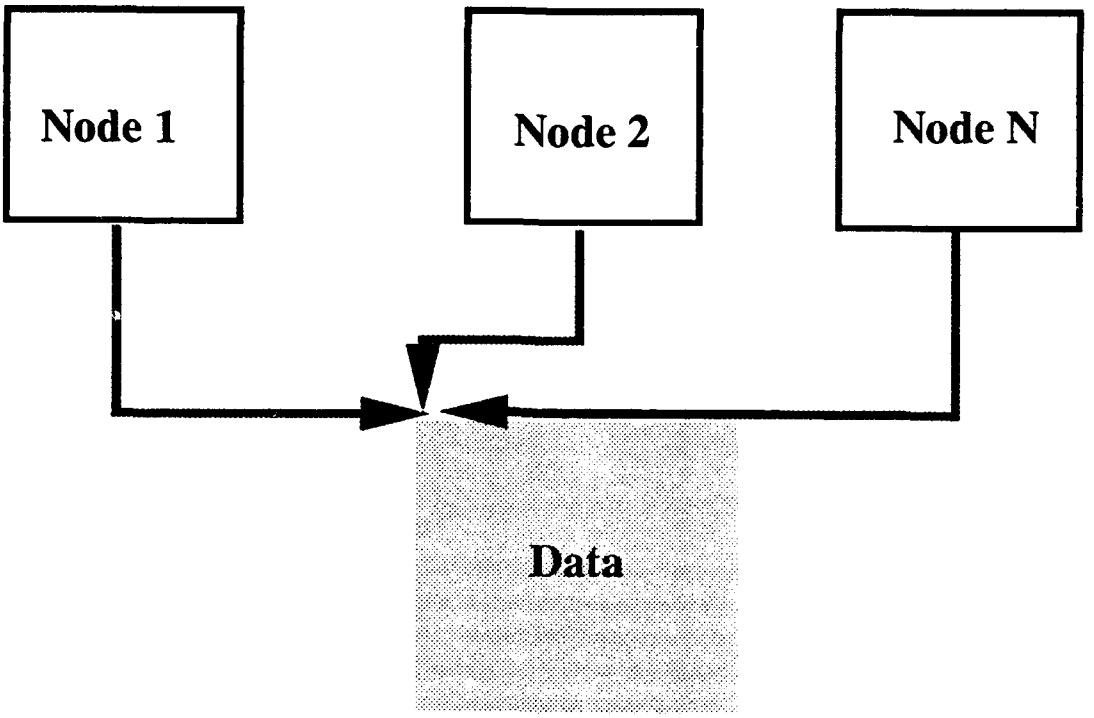


Figure 4. *MRS* nodes sharing data area.

2. NETWORK DISTRIBUTED GLOBAL MEMORY (*NDGM*)

NDGM is a layer of routines on top of *MRS* that frees the application from much of the bookkeeping of message passing. An application communicates with others by writing and reading data into a virtual space. Even though file memory physically resides on several distributed machines, it is accessed as a contiguous data buffer. The *NDGM* library manages the details of accessing this global memory.

The actual layout of the memory is described in a file that is given to the shell script *ndgm_start*. Each line in the file gives a hostname and a size in bytes. Lines that start with "#" are ignored:

```
# NDGM Description File
Node        cpu1.arl.army.mil        5000000
Node        cpu2.arl.army.mil       20000000
Node        cpu3.arl.army.mil       10000000
```

The shell script *ndgm_start* will execute an *rsh* to each of these hosts and start a server program. Once all of the servers have been started, they are all given the mapping of all of the servers in the system. In this way, all of the servers know the assigned starting and end address of every other server. This mapping is also written to the file *ndgm_current_net.dat*.

To use the virtual buffer created by *ndgm_start*, an application makes a call to *ndgm_init()*. This reads the system description from one of the servers and assigns a unique node ID. This ID stays constant for the duration of the application. The system description defines mapping from the global address space to local machine offsets.

Each of the servers started by *ndgm_start* waits for requests from clients to access its data. The server is continuously executing a blocking read, so little CPU time is being consumed. If the server was started by root, it automatically tries to lock its data into core memory so that it does not get swapped out by the operation system. The server will not exit until it receives an *NDGM_TERM* command from a client.

The system can be terminated by running the *ndgm_stop* shell script. All servers started up by *ndgm_start* are sent an *NDGM_TERM* command.

As in Figure 5, *NDGM* sets up an arbitrary size virtual memory array. This memory area is physically distributed across several machines, but is accessed as one continuous memory block. There are routines to put data into global memory and to get data from global memory. These routines access the data as a contiguous block of bytes and not as any particular data type in much the same style as *memcopy()*. This leaves the actual use of the area application defined.

The actual transport of the data is accomplished through *MRS* and is transparent to the application. If the requested area spans several physical machines, the application need not be aware of its layout. The access routines handle all of the necessary message passing.

The physical memory for each block of data is allocated from shared memory. This allows fast access by an application, to a block that is on the same physical machine. Once again, the access routines take care of detecting this situation.

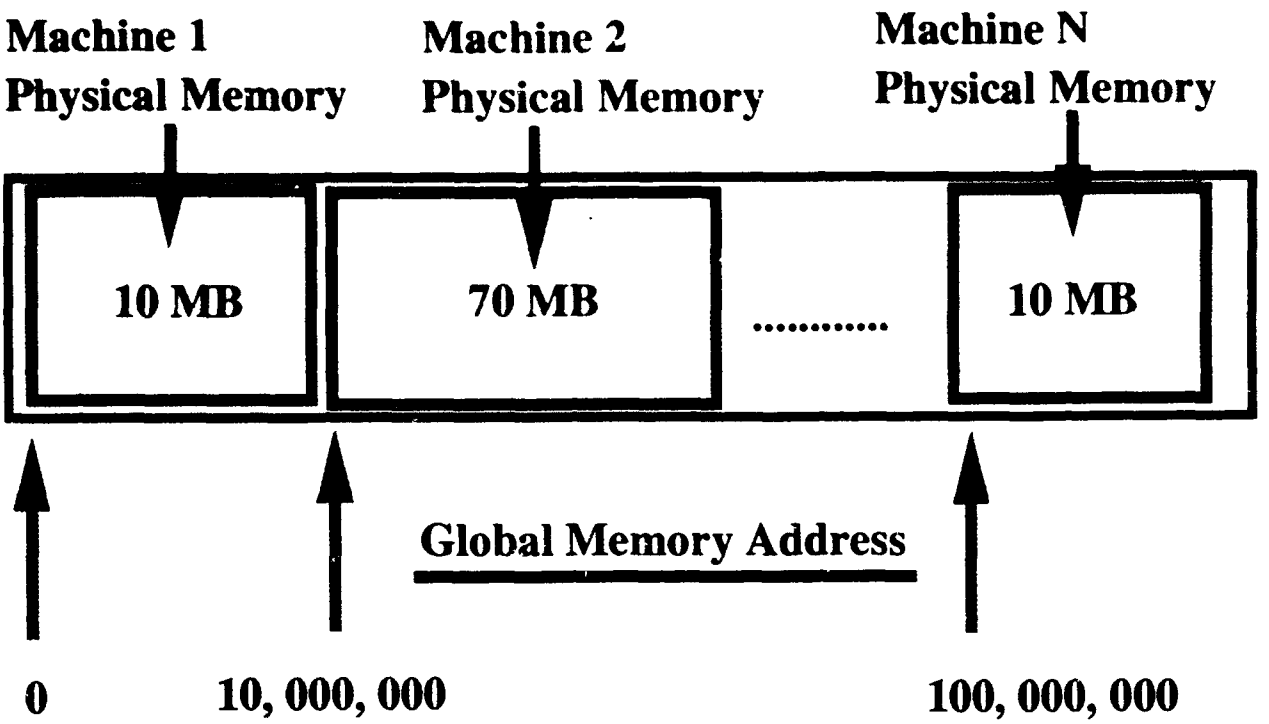


Figure 5. *NDGM* system

In addition to data access routines, *NDGM* contains synchronization mechanisms: memory locks, semaphores, and barriers. These mechanisms are implemented in the *NDGM* server and do not use the associated operating system mechanisms.

Memory locks allow an application to obtain an advisory lock on any section of the global memory. These locks do not restrict access to the data but prevent other locks on the same memory area from being obtained until the original lock is released.

Memory locks are always blocking. This means that a call to lock a section of memory will not return until the entire section has been successfully locked or upon system error. The *NDGM* server maintains a list of pending requests and grants memory locks when the resource becomes available.

Semaphores are implemented in much the same way as memory locks. Only one client can obtain a specified semaphore at any time; all others are blocked until the current owner of the semaphore releases it.

Barriers are used to coordinate activity between several processes. One process sets an initial barrier value. Each process that subsequently checks into the barrier will decrement that value. When the barrier value reaches zero, all processes that have checked in are notified. The barrier value is then automatically reset.

Access to global memory and synchronization mechanisms is accomplished through the following routines:

```
int
ndgm_init(ndgm_node_id, hostname, mrs_host_id, verbose)
int       ndgm_node_id
char      *hostname                          Connect to Global Memory
int       mrs_node_id
int       verbose

int
ndgm_put(address, source, length)
NDGM_ADDR  address
void                    *source      Puts Data Into Global Memory
NDGM_LENGTH             length

int
ndgm_get(address, destination, length)
NDGM_ADDR  address
void                    *destination Gets Data From Global Memory
NDGM_LENGTH             length

int
ndgm_lock(address, length)
NDGM_ADDR  address
NDGM_LENGTH             length       Obtains Memory Lock

int
ndgm_unlock(address, length)
NDGM_ADDR  address
NDGM_LENGTH             length       Unlocks a Section of Global Memory

int
ndgm_sema_get(sema_id)
NDGM_KEY    sema_id                    Obtain Semaphore

void
ndgm_sema_release(sema_id)
NDGM_KEY    sema_id                    Release Semaphore
```

```
int
ndgm_barrier_init(barrier_id, value)
NDGM_KEY    barrier_id
int                         value          Initialize Barrier

int
ndgm_barrier wait(barrier_id)
NDGM_KEY    barrier_id                     Check Into a Barrier

int
ndgm_dump(filename, address, length)
char                        *filename      Dump Memory Image to
NDGM_ADDR address                          File (Parallel I/O)
NDGM_LENGTH                 length

int
ndgm_undump(filename, address, length)
char                        *filename      Read Memory Image From
NDGM_ADDR address                          File (Parallel I/O)
NDGM_LENGTH                 length
```

3. *DZONAL*: AN APPLICATION

Dzonal is a distributed version of "*The Zonal Code*" by Dr. Nishee. Patel (Patel, Sturek, and Smith 1989). The purpose here is not to document the *Dzonal* code but to show how *NDGM* was used to develop a distributed application and utilities.

Dzonal is a full three-dimensional, Navier-Stokes flow solver for supersonic flow. A copy of the *Dzonal* executable is run on multiple machines and coordinated through the use of barriers. Unix shell scripts are used to decompose the computational domain into fairly even chunks and to start the application on the remote machine. There is no explicit message passing. Rather, all coordination is accomplished through *NDGM*.

As an example, assume there is a geometry with two blocks (Figure 6). The first is 20x20x20 and the second is 30x20x30. The two blocks overlap along the I dimension at I=[19,20] of block 1 and I=[1,2] of block 2.

In addition, assume that there are five identical processors on which to distribute the application. Each is a workstation with the same amount of main memory and disk space. Their hostnames are CPU1,

CPU2, CPU3, CPU4, and CPU5. With this arrangement, the layout (chosen by the *Dzonal* domain decomposition utility) might look like Table 1.

Table 1. Domain Decomposition With Identical Processors

cpu1	1	1	20	1	20	1	11
cpu2	1	1	20	1	20	10	20
cpu3	2	1	30	1	20	1	11
cpu4	2	1	30	1	20	10	21
cpu5	2	1	30	1	20	20	30

Notice that there is a 1-K plane overlap among processors. This allows each processor to only compute on interior points; inner-block boundaries are communicated each timestep.

With this layout, the exact amount of global memory is assigned to each processor that will allow its interior points to be assigned to that processor's local memory. For example, CPU4 has K planes 10–21 of block 2. Plane 10 and 21, however, are interior to CPU3 and CPU5, so they are assigned to those CPUs respectively. CPU4 will have K planes 11–20 in its local memory. With a 32-bit floating point number, and assuming there is a need to store 50 values for each grid node (X, Y, Z, Temp. Press ...), CPU4 would be assigned [10planes * 30(i) * 20(j) * 50values * 4 bytes] = 1.2 Megabytes of local memory. This local memory would then be mapped to some global address (CPU1 would start at 0). The global addresses assigned to cpu4 will be called *Add_B2_K11* through *Add_B2_K20*.

For each timestep, CPU4 will calculate values for its interior points and possibly any global boundary conditions (i.e., a wall at J = 1, outflow at I = 30, etc.) then write those values to global memory. Since all of these values are in local memory, this is a fast-writing operation. Once CPU4 has output its values, it waits in a barrier. When all CPUs have checked into the barrier (they have all computed and output their data) they can then read back the information for their boundaries. This includes the cross block communication for the overlap of block 1 and block 2. While this communication scheme might get quite

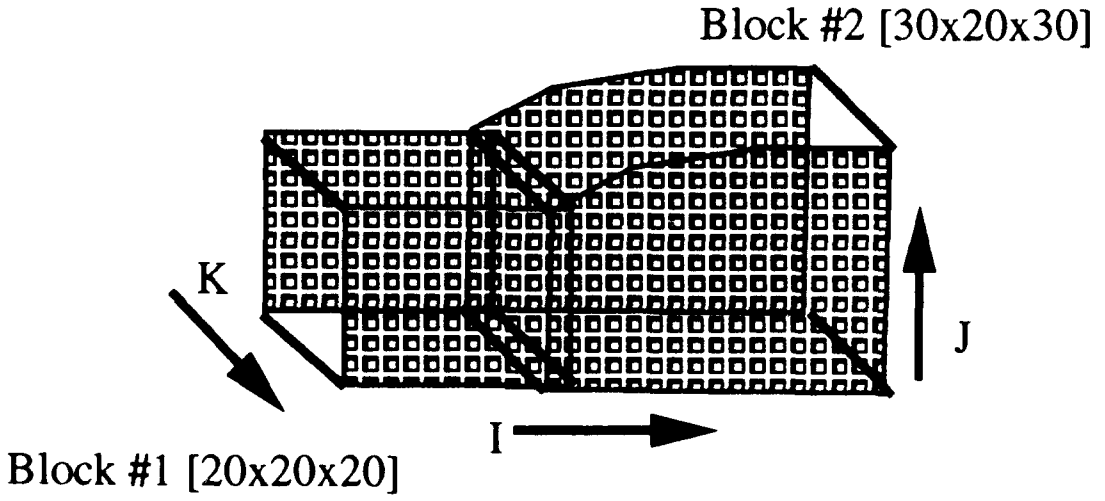


Figure 6. Example *Dzonal* application.

complicated, if explicit message passing is used, it is quite straightforward when viewed as a single shared memory.

Nodes that contain absolute boundaries then apply the appropriate boundary conditions and write that data to global memory. When all nodes check into the final barrier, the application continues to the next timestep. The local to global mapping of this application would look like Figure 7.

Since the *NDGM* servers are separate processes accessed by the *Dzonal* clients, other clients can access the global memory while the solution is developing. This allows quite useful debugging utilities to be developed. Two such utilities that have been developed are *dz_look* and *dz_draw_plane*. *Dz_look* allows the user to look at any value in global memory by entering its IJK value. *Dz_draw_plane* will pull any subsection of any plane (I, J, or K) out of global memory and send it to a network visualization program called *Bop_View* (Clarke 1994). A fully operational system might look like Figure 8.

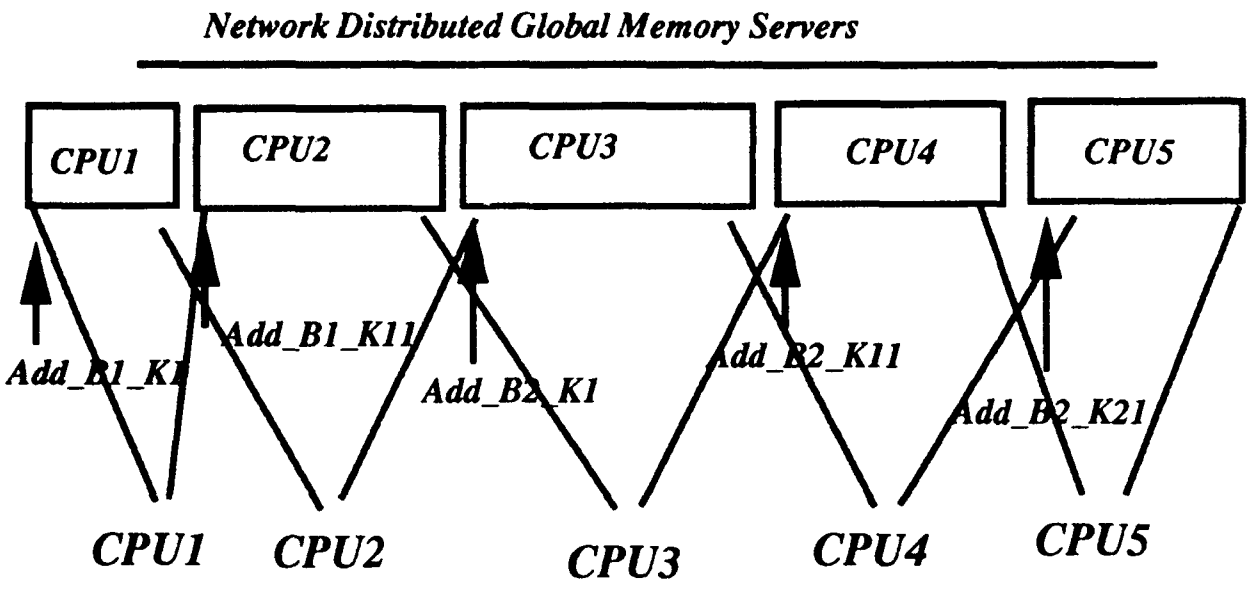


Figure 7. *Dzonal* processes.

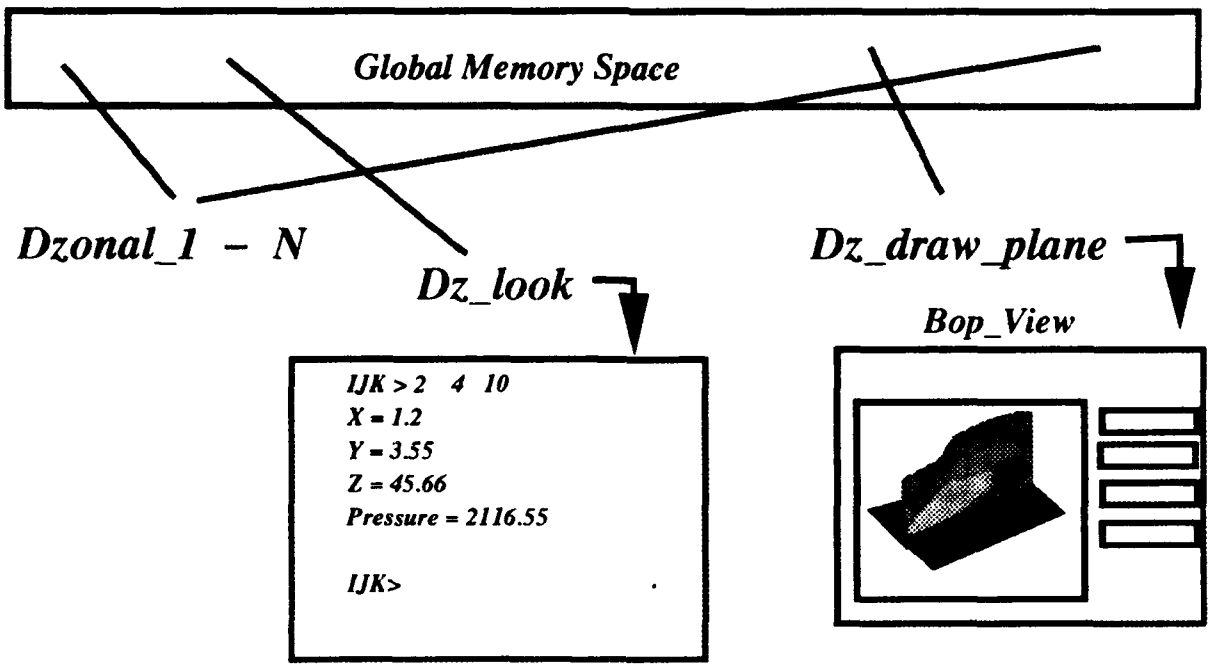


Figure 8. Operational system.

4. CONCLUSIONS

NDGM allows applications to view a physically distributed group of processors as a shared memory parallel machine. Although many factors determine communication speed such as CPU and network load, the following numbers are good "ballpark" numbers for access times:

Transfer to local *NDGM* server:	6 MBytes/second
Transfer to remote *NDGM* server:	4 KBytes/second

These numbers are averages on a network of Silicon Graphics Indigo workstations run during peak and nonpeak hours. They include different transfer lengths and all setup overhead. Your mileage may vary.

Assuming these transfer rates are sufficient, *NDGM* can be used to develop and run parallel applications on networks of relatively low-cost platforms. When the time spent in a batch queue is taken into account, the total wall clock time may be comparable to larger more expensive platforms. The design goal with *NDGM* is to minimize (dollars/grid node) while maintaining an acceptable (wall clock time/grid node).

INTENTIONALLY LEFT BLANK.

5. REFERENCES

Clarke, Jerry. "Remote Data Transfer (RDT): An Interprocess Data Transfer Method for Distributed Environments." BRL-TR-3339, U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD, May 1992.

Clarke, Jerry A. "Distributed Heterogeneous Visualization, *Bop* and *Bop_View*." ARL-CR-172, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, September 1994.

Dykstra, Phillip C. "The BRL CAD Package, An Overview." U.S. Ballistic Research Laboratory, Aberdeen Proving Ground, MD, October 1988.

Muuss, Michael. "Workstations, Networking, Distributed Graphics, and Parallel Processing." U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD, October 1988.

Patel, N., W. Sturek, and G. Smith. "Parallel Computation of Supersonic Flow Using a Three-Dimensional Navier-Stokes Code." BRL-TR-30XX, U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD, November 1988.

"XDR: External Data Representation Standard." RFC-1014, DDN Network Information Center, Menlo Park, CA, June 1987.

INTENTIONALLY LEFT BLANK.

2 Administrator
Defense Technical Info Center
ATTN: DTIC-DDA
Cameron Station
Alexandria, VA 22304-6145

1 Commander
U.S. Army Materiel Command
ATTN: AMCAM
5001 Eisenhower Ave.
Alexandria, VA 22333-0001

1 Director
U.S. Army Research Laboratory
ATTN: AMSRL-OP-SD-TA,
Records Management
2800 Powder Mill Rd.
Adelphi, MD 20783-1145

3 Director
U.S. Army Research Laboratory
ATTN: AMSRL-OP-SD-TL,
Technical Library
2800 Powder Mill Rd.
Adelphi, MD 20783-1145

1 Director
U.S. Army Research Laboratory
ATTN: AMSRL-OP-SD-TP,
Technical Publishing Branch
2800 Powder Mill Rd.
Adelphi, MD 20783-1145

2 Commander
U.S. Army Armament Research,
Development, and Engineering Center
ATTN: SMCAR-TDC
Picatinny Arsenal, NJ 07806-5000

1 Director
Benet Weapons Laboratory
U.S. Army Armament Research,
Development, and Engineering Center
ATTN: SMCAR-CCB-TL
Watervliet, NY 12189-4050

1 Director
U.S. Army Advanced Systems Research
and Analysis Office (ATCOM)
ATTN: AMSAT-R-NR, M/S 219-1
Ames Research Center
Moffett Field, CA 94035-1000

1 Commander
U.S. Army Missile Command
ATTN: AMSMI-RD-CS-R (DOC)
Redstone Arsenal, AL 35898-5010

1 Commander
U.S. Army Tank-Automotive Command
ATTN: AMSTA-JSK (Armor Eng. Br.)
Warren, MI 48397-5000

1 Director
U.S. Army TRADOC Analysis Command
ATTN: ATRC-WSR
White Sands Missile Range, NM 88002-5502

1 Commandant
U.S. Army Infantry School
ATTN: ATSH-WCB-O
Fort Benning, GA 31905-5000

Aberdeen Proving Ground

2 Dir, USAMSAA
ATTN: AMXSY-D
AMXSY-MP, H. Cohen

1 Cdr, USATECOM
ATTN: AMSTE-TC

1 Dir, USAERDEC
ATTN: SCBRD-RT

1 Cdr, USACBDCOM
ATTN: AMSCB-CII

1 Dir, USARL
ATTN: AMSRL-SL-I

5 Dir, USARL
ATTN: AMSRL-OP-AP-L

No. of
Copies Organization

1 Computer Sciences Corporation
ATTN: Dr. David Brown
3160 Fairview Park Dr.
Mail Code 265
Falls Church, VA 22042

Aberdeen Proving Ground

11 Dir, USARL
ATTN: AMSRL-CI, William Mermagen
AMSRL-CI-A, Harold Breaux
AMSRL-CI-AC,
John Grosh
Phillip Dykstra
Jerry Clarke
Deborah Thompson
Jennifer Hare
Eric Mark
Richard Angelini
Kathy Burke
AMSRL-CI-C, Walter Sturek

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number ARL-CR-173 Date of Report October 1994

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)